MATLAB: A Practical Introduction to Programming and Problem Solving

Fifth Edition

PRACTICE PROBLEM SOLUTIONS

Stormy Attaway

College of Engineering
Boston University

Chapter 1

Practice 1.1

Think about what the results would be for the following expressions, and then type them in to verify your answers:

```
>> 1\2
ans =
     2
>> - 5 ^ 2
ans =
   -25
>> (-5) ^ 2
ans =
    25
>> 10-6/2
ans =
     7
>> 5*4/2*3
ans =
    30
```

Practice 1.2

Generate a random

- real number in the range (0,1)

```
rand
```

- real number in the range (0, 100)

```
rand*100
```

- real number in the range (20, 35)

```
rand*(35-20)+20
```

- integer in the inclusive range from 1 to 100

```
randi(100)
```

- integer in the inclusive range from 20 to 35

```
randi([20, 35])
```

Practice 1.3

Think about what would be produced by the following expressions, and then type them in to verify your answers.

```
>> 3 == 5 + 2
ans =
     0

>> 'b' < 'a' + 1
ans =
     0

>> 10 > 5 + 2
ans =
     1

>> (10 > 5) + 2
ans =
     3

>> 'c' == 'd' - 1 && 2 < 4
ans =
     1

>> 'c' == 'd' - 1 || 2 > 4
ans =
     1

>> xor('c' == 'd' - 1, 2 > 4)
ans =
     1

>> xor('c' == 'd' - 1, 2 < 4)
ans =
     0

>> 10 > 5 > 2
ans =
     0
```

Practice 1.4

- Calculate the range of integers that can be stored in the types **int16** and **uint16**. Use **intmin** and **intmax** to verify your results.

```
>> 2^16
ans =
       65536
>> 2^15
ans =
       32768
>> intmin('int16')
ans =
 -32768
>> intmax('int16')
ans =
  32767
>> intmin('uint16')
ans =
      0
>> intmax('uint16')
ans =
65535
```

- Enter an assignment statement and view the type of the variable in the Workspace Window. Then, change its type and view it again. View it also using **whos**.

```
>> clear
>> mynumber = 3*11;
>> whos
  Name          Size             Bytes  Class     Attributes

  mynumber      1x1                  8  double

>> mynumber = int32(mynumber)
mynumber =
  int32
   33
>> whos
  Name          Size             Bytes  Class    Attributes

  mynumber      1x1                  4  int32
```

Practice 1.5

- Find the numerical equivalent of the character 'x'.

- Find the character equivalent of 107.

```
>> double('x')
```

```
  ans =
     120
  >> char(107)
  ans =
      'k'
```

Practice 1.6

Use the **help** function to find out what the rounding functions **fix**, **floor**, **ceil**, and **round** do. Experiment with them by passing different values to the functions, including some negative, some positive, some with fractions less than 0.5 and some greater. *It is very important when testing functions that you thoroughly test by trying different kinds of arguments!*

```
>> help fix
 fix    Round towards zero.
    fix(X) rounds the elements of X to the nearest integers
    towards zero.

    See also floor, round, ceil.

    Reference page for fix
    Other functions named fix

>> help floor
 floor  Round towards minus infinity.
    floor(X) rounds the elements of X to the nearest integers
    towards minus infinity.

    See also round, ceil, fix.

    Reference page for floor
    Other functions named floor

>> help ceil
 ceil   Round towards plus infinity.
    ceil(X) rounds the elements of X to the nearest integers
    towards infinity.

    See also floor, round, fix.

    Reference page for ceil
    Other functions named ceil

>> help round
 round  rounds towards nearest decimal or integer

    round(X) rounds each element of X to the nearest integer.
```

round(X, N), for positive integers N, rounds to N digits to the right
of the decimal point. If N is zero, X is rounded to the nearest integer.
If N is less than zero, X is rounded to the left of the decimal point.
N must be a scalar integer.

round(X, N, 'significant') rounds each element to its N most significant
digits, counting from the most-significant or left side of the number.
N must be a positive integer scalar.

round(X, N, 'decimals') is equivalent to round(X, N).

For complex X, the imaginary and real parts are rounded independently.

Examples

```
% Round pi to the nearest hundredth
>> round(pi, 2)
     3.14

% Round the equatorial radius of the Earth, 6378137 meters,
% to the nearest kilometer.
round(6378137, -3)
     6378000

% Round to 3 significant digits
format shortg;
round([pi, 6378137], 3, 'significant')
     3.14     6.38e+06
```

If you only need to display a rounded version of X,
consider using fprintf or num2str:

```
fprintf('%.3f\n', 12.3456)
     12.346
fprintf('%.3e\n', 12.3456)
     1.235e+01
```

See also floor, ceil, fprintf.

```
    Other functions named round
>>

>> fix(5.2)
ans =
     5
>> fix(5.8)
ans =
     5
>> floor(5.2)
ans =
     5
>> floor(5.8)
ans =
     5
>> ceil(5.2)
ans =
     6
>> ceil(5.8)
ans =
     6
>> round(5.2)
ans =
     5
>> round(5.8)
ans =
     6
>> fix(-5.2)
ans =
    -5
>> fix(-5.8)
ans =
    -5
>> floor(-5.2)
ans =
    -6
>> floor(-5.8)
ans =
    -6
>> ceil(-5.2)
ans =
    -5
>> ceil(-5.8)
ans =
    -5
>> round(-5.2)
ans =
```

```
    -5
>> round(-5.8)
ans =
    -6
>> fix(4.5)
ans =
     4
>> floor(4.5)
ans =
     4
>> ceil(4.5)
ans =
     5
>> round(4.5)
ans =
     5
```

MATLAB: A Practical Introduction to Programming and Problem Solving

Fifth Edition

SOLUTION MANUAL

Stormy Attaway

College of Engineering
Boston University

I. Introduction to Programming Using MATLAB

Chapter 1: Introduction to MATLAB

Exercises

1) Create a variable *myage* and store your age in it. Add 2 to the value of the variable. Subtract 3 from the value of the variable. Observe the Workspace Window and Command History Window as you do this.

```
>> myage = 20;
>> myage = myage + 2;
>> myage = myage - 3;
```

2) Create a variable to store the atomic weight of iron (55.85).

```
>> iron_at_wt = 55.85;
```

3) Explain the difference between these two statements:

```
result = 9*2
result = 9*2;

Both will store 18 in the variable result.  In the first,
MATLAB will display this in the Command Window; in the
second, it will not.
```

4) In what variable would the result of the following expression be stored:

```
>> 3 + 5
```

ans

5) Use the built-in function **namelengthmax** to find out the maximum number of characters that you can have in an identifier name under your version of MATLAB.

```
>> namelengthmax
ans =
     63
```

6) Create two variables to store a weight in pounds and ounces. Use **who** and **whos** to see the variables. Use **class** to see the types of the variables. Clear one of them using **clearvars** and then use **who** and **whos** again.

```
>> clear
```

```
>> pounds = 4;
>> ounces = 3.3;
>> who

Your variables are:

ounces  pounds

>> whos
  Name        Size              Bytes  Class     Attributes

  ounces      1x1                   8  double
  pounds      1x1                   8  double

>> class(ounces)
ans =
    'double'
>> clearvars ounces
>> who

Your variables are:

ans     pounds
```

7) Explore the **format** command in more detail. Use **help format** to find options. Experiment with **format bank** to display dollar values.

```
>> format +
>> 12.34
ans =
+
>> -123
ans =
-
>> format bank
>> 33.4
ans =
         33.40
>> 52.435
ans =
         52.44
```

8) Find a **format** option that would result in the following output format:

```
>> 5/16 + 2/7
ans =
     67/112
```

```
>> format rat
>> 5/16 + 2/7
ans =
      67/112
```

9) Think about what the results would be for the following expressions, and then type them in to verify your answers.

```
     25 / 5 * 3
     4 + 2 ^ 3
     (4 + 1) ^ 2
     2 \ 12 + 5
     4 - 1 * 5
```

```
>> 25 / 5 * 3
ans =
    15
>> 4 + 2 ^ 3
ans =
    12
>> (4 + 1) ^ 2
ans =
    25
>> 2 \ 12 + 5
ans =
    11
>> 4 - 1 * 5
ans =
    -1
```

10) There are 1.6093 kilometers in a mile. Create a variable to store a number of miles. Convert this to kilometers, and store in another variable.

```
     >> miles = 60;
     >> km = miles * 1.6093
     km =
        96.5580
```

11) Create a variable *ftemp* to store a temperature in degrees Fahrenheit (F). Convert this to degrees Celsius (C) and store the result in a variable *ctemp*. The conversion factor is C = (F – 32) * 5/9.

```
>> ftemp = 75;
>> ctemp = (ftemp - 32) * 5/9
ctemp =
   23.8889
```

12) The following assignment statements either contain at least one error, or could be improved in some way. Assume that *radius* is a variable that has been initialized. First, identify the problem, and then fix and/or improve them:

```
33 = number

     The variable is always on the left
     number = 33

my variable =  11.11;

     Spaces are not allowed in variable names
     my_variable = 11.11;

area = 3.14 * radius^2;

     Using pi is more accurate than 3.14
     area = pi * radius^2;

x = 2 * 3.14 * radius;

     x is not a descriptive variable name
     circumference  =  2 * pi * radius;
```

13) Experiment with the functional form of some operators such as **plus**, **minus**, and **times**.

```
>> plus(4, 8)
ans =
    12
>> plus(3, -2)
ans =
     1
>> minus(5, 7)
ans =
    -2
>> minus(7, 5)
ans =
     2
>> times(2, 8)
ans =
    16
```

14) Explain the difference between constants and variables.

```
Constants store values that are known and do not change.
Variables are used when the value will change, or when the value
```

is not known to begin with (e.g., the user will provide the value).

15) Generate a random

- real number in the range (0, 30)

```
rand * 30
```

- real number in the range (10, 100)

```
rand*(100-10)+10
```

- integer in the inclusive range from 1 to 20

```
randi(20)
```

- integer in the inclusive range from 0 to 20

```
randi([0, 20])
```

- integer in the inclusive range from 30 to 80

```
randi([30, 80])
```

16) Get into a new Command Window, and type **rand** to get a random real number. Make a note of the number. Then, exit MATLAB and repeat this, again making a note of the random number; it should be the same as before. Finally, exit MATLAB and again get into a new Command Window. This time, change the seed before generating a random number; it should be different.

```
>> rand
ans =
0.8147

>> rng('shuffle')
>> rand
ans =
0.4808
```

17) What is the difference between x and 'x'?

In an expression, the first would be interpreted as the name of a variable, whereas 'x' is the character x.

18) What is the difference between 5 and '5'?

```
The first is the number 5, the second is the character 5.
(Note: int32(5) is 53.  So, 5+1 would be 6.  '5'+1 would be 54.)
```

19) The combined resistance R_T of three resistors R_1, R_2, and R_3 in parallel is given by

$$R_T = \frac{1}{\frac{1}{R_1}+\frac{1}{R_2}+\frac{1}{R_3}}$$

Create variables for the three resistors and store values in each, and then calculate the combined resistance.

```
>> r1 = 3;
>> r2 = 2.2;
>> r3 = 1.5;
>> rt = 1/(1/r1 + 1/r2 + 1/r3)
rt =
    0.6875
```

20) What would be the result of the following expressions?

`'b' >= 'c' - 1`	1
`3 == 2 + 1`	1
`(3 == 2) + 1`	1
`xor(5 < 6, 8 > 4)`	0

21) Explain why the following expression results in 0 for false:

```
5 > 4 > 1

Evaluated from left to right: 5 > 4 is true, or 1.
1 > 1 is false.
```

22) Explain why the following expression results in 1 for true:

```
result = -20;
0 <= result <= 10

Evaluated from left to right: 0 <= -20 is false, or 0.
Then 0 <= 10 is true, or 1.
```

23) Create two variables *x* and *y* and store numbers in them. Write an expression that would be **true** if the value of *x* is greater than five or if the value of *y* is less than ten, but not if both of those are **true**.

```
>> x = 3;
>> y = 12;
>> xor(x > 5, y < 10)
ans =
     0
```

24) Use the equality operator to verify that 4*10^3 is equal to 4e3.

```
>> 4 * 10 ^ 3 == 4e3
ans =
  logical
   1
```

25) In the ASCII character encoding, the letters of the alphabet are in order: 'a' comes before 'b' and also 'A' comes before 'B'. However, which comes first - lower or uppercase letters?

```
>> int32('a')
ans =
          97
>> int32('A')
ans =
          65

The upper case letters
```

26) Are there equivalents to **intmin** and **intmax** for real number types? Use **help** to find out.

```
>> realmin
ans =
  2.2251e-308
>> realmin('double')
ans =
  2.2251e-308
>> realmin('single')
ans =
  1.1755e-38
>> realmax
ans =
  1.7977e+308
```

27) Use **intmin** and **intmax** to determine the range of values that can be stored in the types **uint32** and **uint64**.

```
>> intmin('uint32')
ans =
           0
>> intmax('uint32')
ans =
  4294967295
>> intmin('uint64')
ans =
                    0
>> intmax('uint64')
ans =
 18446744073709551615
```

28) Use the **cast** function to cast a variable to be the same type as another variable.

```
>> vara = uint16(3 + 5)
vara =
      8
>> varb = 4*5;
>> class(varb)
ans =
double
>> varb = cast(varb, 'like', vara)
varb =
     20
>> class(varb)
ans =
uint16
```

29) Use **help elfun** or experiment to answer the following questions:

- Is **fix(3.5)** the same as **floor(3.5)**?

```
>> fix(3.5)
ans =
     3
>> floor(3.5)
ans =
     3
```

- Is **fix(3.4)** the same as **fix(-3.4)**?

```
>> fix(3.4)
ans =
     3
>> fix(-3.4)
```

```
ans =
    -3
```

- Is **fix(3.2)** the same as **floor(3.2)**?

```
>> fix(3.2)
ans =
     3
>> floor(3.2)
ans =
     3
```

- Is **fix(-3.2)** the same as **floor(-3.2)**?

```
>> fix(-3.2)
ans =
    -3
>> floor(-3.2)
ans =
    -4
```

- Is **fix(-3.2)** the same as **ceil(-3.2)**?

```
>> fix(-3.2)
ans =
    -3
>> ceil(-3.2)
ans =
    -3
```

30) For what range of values is the function **round** equivalent to the function **floor**?
For what range of values is the function **round** equivalent to the function **ceil**?

```
For positive numbers: when the decimal part is less than .5
For negative numbers: when the decimal part is greater than or
equal to .5
```

31) Use **help** to determine the difference between the **rem** and **mod** functions.

```
>> help rem
 rem    Remainder after division.
    rem(x,y) is x - n.*y where n = fix(x./y) if y ~= 0.
    By convention:
       rem(x,0) is NaN.
       rem(x,x), for x~=0, is 0.
       rem(x,y), for x~=y and y~=0, has the same sign as x.
```

```
rem(x,y) and MOD(x,y) are equal if x and y have the same sign,
but differ by y if x and y have different signs.

>> help mod
 mod    Modulus after division.
    mod(x,y) is x - n.*y where n = floor(x./y) if y ~= 0.

    By convention:
       mod(x,0) is x.
       mod(x,x) is 0.
       mod(x,y), for x~=y and y~=0, has the same sign as y.
```

32) Use the equality operator to verify that log10(1000) is 3.

```
>> log10(1000)== 3
ans =
  logical
   1
```

33) Using only the integers 2 and 3, write as many expressions as you can that result in 9. Try to come up with at least 10 different expressions (e.g., don't just change the order). Be creative! Make sure that you write them as MATLAB expressions. Use operators and/or built-in functions.

```
3 ^ 2

2 ^ 3 + (3 - 2)

3 * 3

3 ^ 3 - 3 * 3 * 2

2^3 + abs(2-3)

2^3 + sign(3)

3/2*2*3

2\3*2*3

sqrt(3^(2+2))

nthroot(3^(2+2),2)
```

34) A vector can be represented by its rectangular coordinates x and y or by its polar coordinates r and θ. Theta is measured in radians. The relationship between them is given by the equations:

```
x = r * cos(θ)
y = r * sin(θ)
```

Assign values for the polar coordinates to variables *r* and *theta*. Then, using these values, assign the corresponding rectangular coordinates to variables *x* and *y*.

```
>> r = 5;
>> theta = 0.5;
>> x = r * cos(theta)
x =
    4.3879
>> y = r * sin(theta)
y =
    2.3971
```

35) In special relativity, the Lorentz factor is a number that describes the effect of speed on various physical properties when the speed is significant relative to the speed of light. Mathematically, the Lorentz factor is given as:

$$\gamma = \frac{1}{\sqrt{1 - \frac{v^2}{c^2}}}$$

Use 3×10^8 m/s for the speed of light, *c*. Create variables for *c* and the speed *v* and from them a variable *lorentz* for the Lorentz factor.

```
>> c = 3e8;
>> v = 2.9e8;
>> lorentz = 1 / sqrt(1 - v^2/c^2)
lorentz =
    3.9057
```

36) A company manufactures a part for which there is a desired weight. There is a tolerance of N percent, meaning that the range between minus and plus N% of the desired weight is acceptable. Create a variable that stores a weight, and another variable for N (e.g., set it to 2). Create variables that store the minimum and maximum values in the acceptable range of weights for this part.

```
>> weight = 12.3;
>> N = 2;
>> mymin = weight - weight*0.01*N
mymin =
   12.0540
>> mymax = weight + weight*0.01*N
```

```
mymax =
   12.5460
```

37) A chemical plant releases an amount *A* of pollutant into a stream. The maximum concentration *C* of the pollutant at a point which is a distance *x* from the plant is:

$$C = \frac{A}{x}\sqrt{\frac{2}{\Pi e}}$$

Create variables for the values of *A* and *x*, and then for *C*. Assume that the distance *x* is in meters. Experiment with different values for *x*.

```
>> A = 30000;
>> x = 100;
>> C = A/x * sqrt(2/(pi*exp(1)))
C =
        145.18
>> x = 1000;
>> C = A/x * sqrt(2/(pi*exp(1)))
C =
         14.52
>> x = 20000;
>> C = A/x * sqrt(2/(pi*exp(1)))
C =
          0.73
```

38) The geometric mean g of n numbers x_i is defined as the nth root of the product of x_i:

$$g = \sqrt[n]{x_1 x_2 x_3 \ldots x_n}$$

(This is useful, for example, in finding the average rate of return for an investment which is something you'd do in engineering economics). If an investment returns 15% the first year, 50% the second, and 30% the third year, the average rate of return would be $(1.15*1.50*1.30)^{1/3}$.) Compute this.

```
>> x1 = 1.15;
>> x2 = 1.5;
>> x3 = 1.3;
>> gmean = nthroot(x1*x2*x3, 3)
gmean =
          1.31
```

39) Use the **deg2rad** function to convert 180 degrees to radians.

```
>> deg2rad(180)
ans =
    3.1416
```